Universidad Autonoma del Estado de Hidalgo
Licenciatura en Ciencias Computacionales
Automatas y Compiladores
Docente: Dr.Eduardo Cornejo Velazquez

REPORTE DE PRÁCTICA NO.1:

Análisis de Expresiones Regulares

ALUMNO:
Naomi Romero López

1. Introducción

El análisis de cadenas de símbolos es una actividad fundamental en el área de las Ciencias Computacionales, especialmente en el diseño de compiladores, intérpretes y sistemas de procesamiento de lenguajes. Dicho análisis se basa en los conceptos de lenguajes formales y gramáticas, los cuales permiten definir de manera precisa la estructura y las reglas que deben cumplir las palabras que conforman un lenguaje.

Una gramática formal proporciona la manera de generar un lenguaje a partir de un conjunto finito de símbolos, mediante reglas de producción bien definidas. En particular, las gramáticas regulares (Tipo 3) permiten generar lenguajes regulares, los cuales pueden ser descritos de forma compacta y eficiente mediante expresiones regulares. Estas constituyen una herramienta poderosa para identificar, validar y clasificar cadenas de símbolos de acuerdo con patrones específicos.

En esta práctica se aborda el análisis de expresiones regulares como una estrategia para reconocer diferentes tipos de palabras dentro de un lenguaje regular, tales como números enteros, palabras en mayúsculas y minúsculas, identificadores y símbolos. Para ello, se implementan programas en un lenguaje de programación de alto nivel, donde se aplican algoritmos básicos y expresiones regulares con el fin de comprender la función de un analizador léxico dentro del compilador.

El desarrollo de esta practica ha permitido emplear conceptos teóricos de lenguajes formales en un contexto practico, ayudando a fortalecer habilidades de programación, análisis de patrones y el uso de herramientas formales propias del análisis léxico, encargado de reconocer, clasificar y validar los componentes básicos de un programa dentro del proceso de compilación.

2. Marco teórico

Lenguajes formales

Los lenguajes formales constituyen una herramienta fundamental en las Ciencias Computacionales para describir conjuntos de cadenas de símbolos generadas a partir de reglas bien definidas. Un lenguaje formal se define como un conjunto de cadenas construidas sobre un alfabeto finito, el cual permite modelar de manera matemática la sintaxis de los lenguajes de programación y otros sistemas computacionales.

El estudio de los lenguajes formales proporciona las bases teóricas necesarias para el análisis automático de programas, ya que permite determinar si una cadena pertenece o no a un lenguaje específico, lo cual es esencial dentro del proceso de compilación. [4]

Gramáticas Formales

Una gramática formal es un mecanismo matemático utilizado para generar lenguajes formales mediante reglas que definen la estructura y validez de una cadena de caracteres aceptadas en un lenguaje formal. Una gramática formal se define como una cuádrupla compuesta por símbolos terminales, símbolos no terminales, un símbolo inicial y un conjunto de producciones.

Las gramáticas permiten describir la estructura sintáctica de un lenguaje de manera precisa y no ambigua. Estas gramáticas se clasifican, de acuerdo con la jerarquía de Chomsky, en cuatro tipos, siendo las gramáticas regulares las más simples y las que se emplean directamente en el análisis léxico. [5]

Lenguajes Regulares y Gramáticas Regulares

Los lenguajes regulares son aquellos que pueden ser generados por gramáticas regulares o reconocidos por autómatas finitos. Este tipo de lenguajes resulta especialmente relevante en la construcción de compiladores, ya que los componentes léxicos de un lenguaje de programación pueden expresarse mediante reglas regulares.

Las gramáticas regulares presentan restricciones en la forma de sus producciones, lo que facilita su implementación computacional y permite el uso de modelos equivalentes como los autómatas finitos deterministas y las expresiones regulares.[6]

Expresiones Regulares

Las expresiones regulares son una secuencia de caracteres utilizada para delimitar patrones de búsqueda. Estas expresiones permiten representar patrones de cadenas mediante operadores como la concatenación, la unión y el cierre de Kleene.

En el ámbito de los compiladores, las expresiones regulares se emplean para definir los patrones que identifican tokens como números, identificadores y símbolos especiales. Su uso simplifica la implementación del análisis léxico y permite un reconocimiento eficiente de cadenas válidas [5]

Análisis léxico

El análisis léxico es la primera etapa del proceso de compilación y tiene como objetivo transformar una secuencia de caracteres agrupados en lexemas en una secuencia de tokens. El analizador léxico se encarga de eliminar elementos irrelevantes como espacios en blanco y comentarios, además de detectar errores léxicos.

Este proceso se apoya principalmente en lenguajes regulares y expresiones regulares para definir formalmente los patrones que describen cada tipo de token. Una correcta implementación del análisis léxico es esencial para garantizar el correcto funcionamiento de las fases posteriores del compilador.[6]

Tokens

Los tokens representan las unidades básicas reconocidas por el analizador léxico. Estos incluyen identificadores, constantes numéricas, palabras reservadas y símbolos del lenguaje.

Cada uno de estos elementos puede describirse mediante una expresión regular específica, lo que permite su reconocimiento automático. Esta clasificación es fundamental para estructurar correctamente el código fuente y facilitar su análisis sintáctico posterior.[4]

3. Herramientas empleadas

Para esta practica fueron utilizados los siguientes recursos de software:

1. Lenguaje de programación Java
2. IDE / Editor: Apache Netbeans
3. Software de apoyo: regex101

4. Objetivos

Objetivo general

Construir programas en un lenguaje de programación de alto nivel que realicen el análisis de cadenas de símbolos que forman palabras basado en la implementación de estructuras de datos y algoritmos de análisis, además del uso de expresiones regulares como segunda estrategia de implementación.

Objetivos específicos

1. Identificar el proceso básico de análisis de cadenas de símbolos de un alfabeto.
2. Implementar estructuras de datos y algoritmos de análisis de cadenas de símbolos en un lenguaje de programación de alto nivel.
3. Implementar expresiones regulares para el análisis de cadenas de símbolos en un lenguaje de programación de alto nivel.

5. Desarrollo

Análisis de Cadenas Mediante Estructuras de Datos

En una primera etapa, se utilizó el lenguaje de programación Java para implementar un programa basado en algoritmos básicos que permitiera analizar cadenas de símbolos de manera secuencial. El objetivo principal de esta fase fue identificar distintos tipos de palabras sin emplear expresiones regulares, con el fin de comprender el proceso básico del análisis léxico. El programa desarrollado permitió reconocer las siguientes categorías:

1. Números enteros
2. Palabras en minúsculas
3. Palabras en mayúsculas
4. Identificadores (Nombres de variables)

Para lograrlo, se definieron algoritmos basicos que permitieron almacenar y recorrer cada carácter de la cadena de entrada. A partir de este recorrido, se aplicaron condiciones lógicas para validar si los símbolos pertenecían a un conjunto específico del alfabeto, determinando así la categoría correspondiente de cada palabra analizada. El código fue implementado en el lenguaje Java. Debido a su extensión, el código fuente completo se encuentra alojado en el siguiente repositorio de GitHub: Primer código en este enlace

Definición del Alfabeto y Expresiones Regulares

Posteriormente, se definió formalmente el alfabeto de símbolos utilizado en el análisis, considerando letras mayúsculas, letras minúsculas, dígitos y símbolos utilizados en el lenguaje de programación Java. A partir de este alfabeto, se diseñaron expresiones regulares capaces de describir los patrones correspondientes a cada tipo de palabra de un lenguaje regular. Las expresiones regulares siguientes permitieron identificar:

1. Números enteros: [0-9]+
2. Palabras en minúsculas: [a-z]+
3. Palabras en mayúsculas: [A-Z]+
4. Identificadores (Nombres de variables): `a-zA-Z_$]`a-zA-Z0-9]*—
5. Símbolos: `[+\-*/%<>=!&^~(){}\[\];,.:?_#@]+`

Análisis y Clasificación Mediante Expresiones Regulares

En la última etapa del desarrollo, se implementó un programa que utilizó expresiones regulares como principal estrategia para el análisis de cadenas de símbolos. El programa permitió leer una cadena de entrada, analizarla mediante las expresiones regulares previamente definidas y clasificar cada palabra según su tipo. El código fue implementado en el lenguaje Java. Debido a su extensión, el código fuente completo se encuentra alojado en el siguiente repositorio de GitHub:Primer código en este enlace

5. Resultados

Se realizaron pruebas con el código desarrollado para verificar el reconocimiento de los operadores y símbolos especiales.

En la Figura 1 se observa el comportamiento del programa ante la opción de solo letras minúsculas. En la

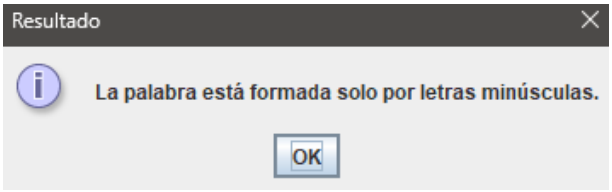

Figure 1: Reconocimiento palabras en minúsculas

Figura 2 se observa el comportamiento del programa ante la opción de identificadores. En la Figura 3 se

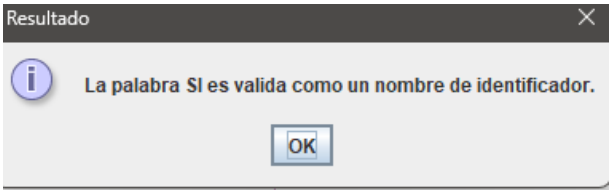

Figure 2: Reconocimiento palabras en minúsculas

observa el resultado del programa con todas las opciones implementando expresiones regulares.

```
run:
123 -> Número entero
hola -> Palabra en minúsculas
JAVA -> Palabra en mayúsculas
contador -> Palabra en minúsculas
_total -> Identificador válido
9variable -> No reconocido
Sumal -> Identificador válido
$%& -> Compuesto por simbolos
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 3: Reconocimiento palabras en minúsculas

Conclusiones

Con la realización de la Práctica 1: Análisis de Expresiones Regulares fue posible comprender cómo se analizan cadenas de símbolos dentro de un lenguaje formal y su relación con el funcionamiento de un compilador. A través de la práctica, se aplicaron conceptos teóricos de manera clara y práctica.

El uso de estructuras de datos permitió entender el proceso básico del análisis léxico, ya que se analizó cada carácter de las cadenas para identificar números enteros, palabras en mayúsculas, palabras en minúsculas e identificadores. Este método ayudó a conocer cómo se puede clasificar la información sin utilizar expresiones regulares.

Por otro lado, el uso de expresiones regulares facilitó el reconocimiento y la clasificación de las cadenas de símbolos, haciendo el proceso más simple y eficiente. Gracias a este enfoque fue posible identificar y contar las diferentes categorías de palabras de forma automática.

La comparación entre ambos enfoques permitió observar que, aunque el análisis manual con estructuras de datos ayuda a comprender mejor el proceso, las expresiones regulares son una herramienta más práctica para implementar analizadores léxicos en situaciones reales.

En conclusión, esta práctica fortaleció los conocimientos sobre lenguajes formales y análisis léxico, y permitió comprender la importancia de las expresiones regulares en el reconocimiento de los elementos básicos de un lenguaje de programación.

Referencias Bibliográficas

References

[1] Brookshear, J. G. (**1995**). *Teoría de la computación: Lenguajes formales, autómatas y complejidad* (1.ª ed.). Addison-Wesley Iberoamericana. ISBN: 978-0201408441.

[2] Giró, J., Vázquez, J., Meloni, B., & Constable, L. (**2015**). *Lenguajes formales y teoría de autómatas*. Alfaomega. ISBN: 978-6077071638.

[3] Ruiz Catalán, J. (**2010**). *Compiladores: teoría e implementación*. Alfaomega. ISBN: 978-6077854385.

[4] Brookshear, J. G. (**1995**). *Teoría de la computación: Lenguajes formales, autómatas y complejidad* (1.ª ed.). Addison-Wesley Iberoamericana. ISBN: 978-0201408441.

[5] Giró, J., Vázquez, J., Meloni, B., & Constable, L. (**2015**). *Lenguajes formales y teoría de autómatas*. Alfaomega. ISBN: 978-6077071638.

[6] Ruiz Catalán, J. (**2010**). *Compiladores: teoría e implementación*. Alfaomega. ISBN: 978-6077854385.

[7] Carraza Sahagún, D. U. (**2024**). *Compiladores: fases de análisis*. Editorial Transdigital.

[8] Chowdhary, K. R. (**2025**). *Theory of Computation: Automata, Formal Languages, Computation and Complexity*. Springer Nature Singapore.